

# Introduction to Computational Text Analysis



American Political Science Association  
Migration and Citizenship Pre-Conference  
Hosted by the UBC Centre For Migration Studies

**Tom Einhorn**

**September 2025**

**With help from Laura Nelson, UBC prAxIs ([ubcecon.github.io/praxis-ubc/](https://ubcecon.github.io/praxis-ubc/))**

## Traditional Qualitative Analysis

Traditional qualitative analysis often involves hand coding text. These methods work well but are slow and require expert training. They are also not scalable.

Distributing hand coding tasks among a team of coders can speed up the process and enable analysis of larger text corpora, but introduces problems:

- Recruiting and training coders
- More expensive
- Inter-coder reliability problems.



# Computational Text Analysis

Computational text analysis **leverages pattern recognition capabilities of machines to automate some qualitative analysis tasks**, such as classification and grouping of text, freeing researchers to focus on interpretive tasks.



- **Scalability:** Enables analysis of large corpora beyond human capacity.
- **Pattern Discovery:** Uncovers latent structures and relationships not easily recognized by humans.
- **Quantification:** Provides numerical measures of abstract concepts (e.g., sentiment, stance).
- **Reproducibility:** Offers systematic, repeatable methods for text analysis.

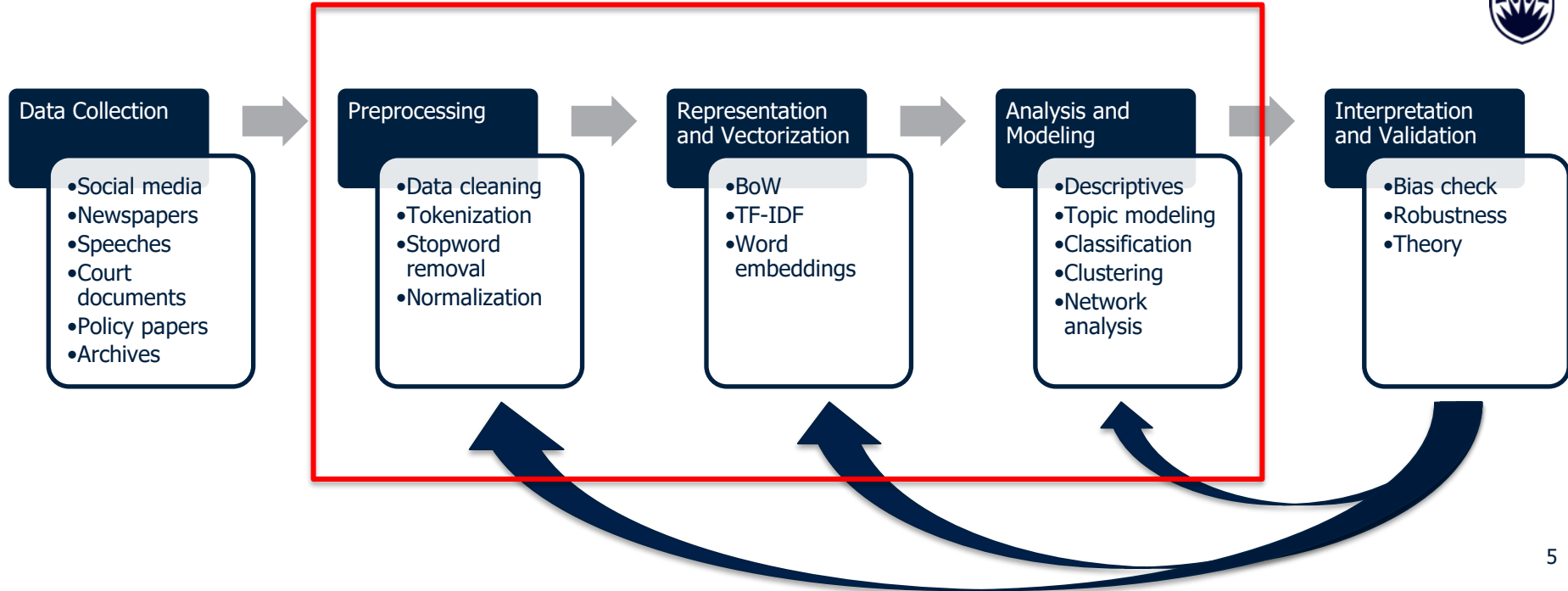
## Session Overview

- First part (~60 minutes)
  - Review core concepts and methodologies
  - Provide examples
  - Progress from simple to more complex
- Second part (~30 minutes)
  - Live demonstration of zero-shot text classification using large language models



Please feel free to ask questions!

# Computational Text Analysis Workflow



# Preprocessing

Before we can work with text, we need to **preprocess** it. The purpose of preprocessing is to **clean and standardize text** so it could be effectively processed by computational models.



Older models require heavy preprocessing because they treat words as isolated tokens and do not handle variation well. Preprocessing for these models thus isolates words, removing noise but also context.

Newer models require less heavy preprocessing, preserving context present in raw text thus allowing them to capture more nuanced meanings.

# Preprocessing

Common preprocessing steps include:

- Tokenization
- Text normalization
- Stop word removal
- Vectorization

Less commonly, preprocessing might also include:

- Stemming
- Lemmatization



# Tokenization

A **token** is a fundamental unit of text, often a word or a subword, created by segmenting larger strings of text.



**Sentence:** The pack of chihuahuas chased after the frightened cat.

**Tokens:** ['The', 'pack', 'of', 'chihuahua', '##s', 'chased', 'after', 'the', 'frightened', 'cat', '.']

**String of tokens:** The pack of chihuahua ##s chased after the frightened cat .

*This example uses bert-based-uncased autotokenization.*



## Stop Words

A **stopword** is an extremely common word in natural language (“the”, “are”, “a”, “is”, etc.). As these words carry little to no meaningful information *on their own*, they are often removed during preprocessing. This reduces noise, allowing models to focus on more meaningful words.



**Original Tokens:** ['The', 'pack', 'of', 'chihuahua', '##s', 'chased', 'after', 'the', 'frightened', 'cat', '.']

**Tokens (stopwords removed):** ['pack', 'chihuahua', '##s', 'chased', 'frightened', 'cat', '.']

# Normalization, Stemming and Lemmatization

These steps are used to **standardize text and reduce noise** so that word variation doesn't confuse the model.



**Text normalization** involves lowercasing, expanding contractions ("I'm" → "I am"), and removing punctuation, special characters and numbers.

**Stemming** means reducing words to root form ("stems") by removing affixes.

Example: "chasing" → "chase"; "frightened" → "frighten"; "chihuahuas" → "chihuahua".

**Lemmatization** means reducing words to dictionary base forms, called lemmas.

Example: "chased" → "chase"; "better" → "good".

# Vectorization

Computational models are fundamentally mathematical machine learning algorithms and require numerical input. Vectorization is the process of **converting tokens into numerical representations** so that models could interact with them. Different vectorization techniques produce different vectors, allowing for different kinds of interpretations.



We will discuss three common vectorization techniques, rising in level of complexity:

- Bag-of-Words (BoW)
- Term Frequency-Inverse Document Frequency (TF-IDF)
- Word embeddings

## Bag-of-Words (BoW)

The simplest way to represent text in numerical values is to **count words**. With BoW, documents are converted into vectors where each position represents how many times a word appears in the document. BoW vectorization represents text as a set (or “bag”) of words, ignoring grammar and word order but preserving word frequency.



**Sentence:** The pack of chihuahuas chased after the frightened cat.

**BoW vector:** [2, 1, 1, 1, 1, 1, 1, 1]

## Bag-of-Words (BoW)

With multiple documents, BoW vectors represent shared vocabulary.



**Sentence I:** “The pack of chihuahuas chased after the frightened cats.”

**Sentence II:** “Chihuahuas are more annoying than cats”

	the	pack	of	chihuahuas	chased	more	after	than	frightened	cats	are	annoying
Sen. I	2	1	1	1	1	0	1	0	1	1	0	0
Sen. II	0	0	0	1	0	1	0	1	0	1	1	1

Sentence I vector: [2, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0]

Sentence II vector: [0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1]

# Bag-of-Words (BoW)

## Strengths:

- Resulting vectors are extremely easy to interpret



## Limitations:

- Vectors become very sparse with large datasets (inefficient, risk of overfitting)
- Context agnostic: "The dog chased after the cat" has the same representation as "the cat chased after the dog".
- Every word is treated equally, so common words dominate even after stop word removal.

```
BoW matrix::
      did  emma  like  little  man   mr  mrs  said  time  whale
Emma    352   865   200    359  235  1153  699   484   279     0
Moby Dick 258     0   647    249  527    63   13   304   334  1226
```

## TF-IDF Vectorization

Term Frequency-Inverted Document Frequency (TF-IDF) **balances word frequency with how *important* a word is** across a collection of documents. The idea is that words common in all documents (like “the”) are less informative, that is – don’t tell us what the document is about. However, words that appear less frequently across all documents but more frequently in a few documents (like “immigration”) are more distinctive, providing information as to what the document is about.



$$\text{TF-IDF} = \underbrace{(\text{how often the word appears in a document})}_{\text{Term Frequency}} \times \underbrace{(\text{how rare the word is across all documents})}_{\text{Inverted Document Frequency}}$$

# TF-IDF Vectorization

## Paragraph I

Immigration plays an important role in shaping the national labor market. Many immigrants contribute to economic growth by filling shortages in critical industries such as agriculture and healthcare. Policymakers often debate whether immigrant workers place downward pressure on wages or whether they enhance productivity by complementing domestic labor. The overall economic impact of immigration depends on how new workers are integrated into the broader workforce.

## Paragraph II

borders enforcement entry surveillance  
detention unauthorized  
enforcement illegal  
security  
border





## Example: TF-IDF With LGBTQ Facebook Posts

**Post** (first 200 characters):

“One Colorado is accepting applications for our fulltime policy manger position. Please share with your networks! Learn more: [URL] #jobs #internship #opportunities #Denver #nonprofitjobs #nonprofit #hiring #HiringColorado ...”

**Preprocessed post** (first 200 characters):

“one colorado accepting applications fulltime policy manger position please share with your networks learn more jobs internship opportunities denver nonprofitjobs nonprofit hiring hiringcolorado coloradono...”

Rank	Term	TF-IDF Score
1	hiring	0.3456
2	denver	0.337
3	accepting	0.332
4	nonprofit	0.3296
5	position	0.3263
6	applications	0.3242
7	opportunities	0.3131
8	colorado	0.2923
9	policy	0.2518
10	share	0.215

## Example: TF-IDF With LGBTQ Facebook Posts

### Post I (first 100 chars):

'colorado accepting applications fulltime policy manger position please share with your networks lear...'

### Post II (first 100 chars):

'have been thinking about sponsoring child fall camp help time this great needed empowering camp want...'

	Doc. I	Score	Doc. II	Score
1	hiring	0.3456	camp	0.5126
2	denver	0.337	scholarship	0.3705
3	accepting	0.332	great	0.302
4	nonprofit	0.3296	cost	0.2276
5	position	0.3263	help	0.213
6	applications	0.3242	fall	0.2105
7	opportunities	0.3131	short	0.2056
8	colorado	0.2923	needed	0.1988
9	policy	0.2518	possible	0.1974
10	share	0.215	send	0.1905

## Example: TF-IDF With LGBTQ Facebook Posts

### Post I:

“one colorado accepting applications fulltime  
policy manger position please share with your  
networks learn more jobs internship opportunities  
denver nonprofitjobs nonprofit hiring  
hiringcolorado coloradono...”

### Actual vector (first 20 positions):

[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,  
**0.3320**, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,  
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,  
0.0000, 0.0000, ...]

	Term	Score	Vector Position
1	hiring	0.3456	634
2	denver	0.337	347
3	accepting	0.332	7
4	nonprofit	0.3296	920
5	position	0.3263	1025
6	applications	0.3242	84
7	opportunities	0.3131	947
8	colorado	0.2923	259
9	policy	0.2518	1018
10	share	0.215	1228

# TF-IDF Vectorization

## Strengths:

- Resulting vectors remain highly interpretable
- Automatically reduces dominance of common words
- Better at identifying distinctive and characteristic terms for each document
- More nuanced representation than simple word counts while maintaining simplicity



## Limitations:

- Vectors become very sparse with large datasets (inefficient, risk of overfitting)
- Context agnostic: "The dog chased after the cat" has the same representation as "the cat chased after the dog"
- Results are better the more documents we have
- More computationally expensive than simple word counts due to IDF calculations

## Supervised vs. Unsupervised Learning

In **supervised learning**, a model is trained on a dataset of documents that are already labeled by humans. During training, the model learns the statistical relationships between features of the text (linguistic patterns, structures, semantic relationships) and assigned labels. Once trained, the model can then predict the labels of new, unseen documents.



## Supervised vs. Unsupervised Learning

In **unsupervised learning**, models discover hidden patterns and structures in text without learning from human-provided labels or categories. While supervised learning requires expensive manual annotation and is limited to predefined categories that may not capture unexpected patterns in the data, **unsupervised approaches automatically identify latent relationships, clusters, and themes** that emerge from the text itself.

Unsupervised approaches are commonly used to discover unknown themes or when labeled training data is unavailable or prohibitively expensive to create.



## Topic Modeling

**Topic modeling** is an unsupervised machine learning method that automatically discovers latent themes or "topics" within a collection of documents. The algorithm identifies patterns in the corpus based on features such as word co-occurrence, semantic relationships, or other metrics. Each document is then characterized based on these patterns. The algorithm groups similar documents together into **clusters** while simultaneously identifying the thematic content that defines each cluster. This is done automatically, without human guidance. Humans then interpret clusters as **topics**.



# Topic Modeling

Consider these sentences<sup>1</sup>:

- French cuisine is some of the best in the world, despite the often lack of spices.
- I've always loved Japanese food, mainly because I prefer fish raw rather than cooked.



What topic would we assign each of these sentences?

<sup>1</sup> Examples from Mattingly, William. 2021. *Introduction to Topic Modeling and Text Classification*. [topic-modeling.pythonhumanities.com](http://topic-modeling.pythonhumanities.com).



# Topic Modeling

Now consider these sentences<sup>1</sup>:

- French cuisine is some of the best in the world, despite the often lack of spices.
- Dogs are very loud and unhygienic, yet people still love having them around.

What topic would we assign each of these sentences?



<sup>1</sup> Examples from Mattingly, William. 2021. *Introduction to Topic Modeling and Text Classification*. [topic-modeling.pythonhumanities.com](http://topic-modeling.pythonhumanities.com).

# Topic Modeling

Topics are **corpus-specific**.

Topics depend on the content of each document, but also on the content of documents in the same corpus.



Topic modeling algorithms identify these natural groupings of similar documents and creates **clusters**. Humans then interpret these clusters and assign labels, creating **topics** out of clusters.

## Topic Modeling

**Hard clustering** algorithms assign each document to a single topic. Documents can be about food or pets, but not both.



**Soft clustering** algorithms allow a single document to belong to multiple topics with varying degrees of membership. A document could be 75% about food and 25% about pets. This is particularly useful with text containing multiple overlapping themes. A policy paper, for instance, could be about climate change *and* about economic policy rather than requiring it to be in one or the other.

# Word Embeddings

**Word embeddings are a vectorization algorithm that takes context into account. Unlike BoW or TF-IDF, which are context-insensitive, word embeddings capture the contextual meaning of words rather than just word frequency or presence.**



Consider these two sentences:

- The film was incredible
- The movie was amazing

These sentences convey the same meaning but use different words to do it. BoW and TF-IDF approaches would thus conclude that they are completely unrelated.

A word embedding model would notice that the words "film" and "movie" sit closely in **embedding space**, as well as the words "incredible" and "amazing", concluding that these sentences are semantically similar.

# Word Embeddings

Now consider these two sentences:

- I deposited money in my bank account
- I am sitting on the river bank



These sentences use the same word - "bank" - to mean different things. As BoW and TF-IDF approaches are context-insensitive, they are unable to distinguish between different meanings of the same word based on context.

Word embedding models solve the problem of context by **learning the contextual relationships between words**. The model examines how words are used across multiple contexts, paying attention to words that appear near each other. Words that appear in similar contexts – meaning, surrounded by similar neighboring words – end up with similar vector representations.

# Word Embeddings

Like BoW or TF-IDF vectors, embedding vectors are also lists of numbers.

**Embedding vectors, however, don't capture word presence or frequency, but semantic relationships to other words.** These

relationships are highly abstract and not directly interpretable by humans.

The model discovers these semantic dimensions automatically by analyzing patterns in how words are used.

Contextual attention enables word embeddings to handle words with multiple meanings. It also enables them to discover words that have similar semantic meanings, like "happy" and "joyful." Embeddings may therefore be used for finding synonyms, measuring text similarity based on meaning rather than word overlap, and understanding semantic relationships in text.



# Word Embeddings

## Strengths:

- Able to detect semantic similarity even when documents use different vocabularies.
- Able to handle synonym variation, unlike older methods. Useful for real world text, where ideas are expressed using different vocabularies.
- Able to comprehend analogous relationships (“man”: “king”; “woman”: [BLANK])

## Limitations:

- Sometimes difficult to understand why models detect similarity.
- Pre-trained models may not work well for specialized vocabularies
- Requires more computational resources and technical expertise compared to older approaches
- Research has shown that models often perpetuate social or cultural biases present in training text corpora



# Word2Vec

**Word2Vec** is one of the most popular word embedding models. It learns word representations by sliding a "window" across text and teaching the model to predict word relationships. The model uses one of two basic architectures:



- **Continuous Bag-of-Word (CBOW):** Predict a center word from surrounding context. Trained on "the cat sat on the mat", the model tries to answer the question: when I see the words "cat", "sat", "the", "mat", what word is likely to be in the center?
- **Skip-Gram:** Predict surrounding word from center word. Trained on "the cat sat on the mat", the model tries to answer the question: when I see the word "cat", what words are likely to appear around it?



## Word2Vec

Word2Vec produces **static embeddings**, meaning each word in the vocabulary gets only one fixed vector representation.



Trained on the sentences “I deposited money in my bank account” and “I am sitting on the river bank”, the model would provide only one vector for the word “bank.” This vector will average out the semantic relationships of the word across the the two sentences.

While this approach captures many semantic relationships effectively, it does not distinguish between different meanings of words based on context.

# BERT

**BERT** (Bidirectional Encoder Representations from Transformers) is an embeddings model that produces **dynamic embeddings**, allowing the same word to have different vector representations depending on context.



BERT uses a transformer architecture that looks at entire sentences simultaneously, analyzing how each word relates to every other word in the context. The model reads each sentence from both directions (bidirectionally), meaning it examines context by looking at words that came before and after each target word.

# BERT

With BERT's dynamic embeddings, the word "bank" receives different vector representations in different contexts:

- "I deposited money in my bank account"  
seeing words like "money" and "account" nearby, "bank" gets a finance-related vector.
- "I sat by the river bank"  
considering words indicating location, "bank" gets a geography-related vector

This allows BERT to handle words with multiple meanings better than static embedding models.



# Large Language Models (LLMs)

**Large Language Models (LLMs)** are systems trained on massive amounts of text to both understand and generate human-like language. Unlike previous models that were designed for specific tasks like identifying topics or parsing grammar, LLMs are general-purpose language systems that can **perform a wide variety of text-based tasks without being specifically trained for each one.**

The word "large" refers to both the enormous amount of text they're trained on and their internal complexity.



## Why BERT Matters for LLMs

Before BERT, researchers had to use huge labeled corpora to train models for specific tasks. The model had to see labeled text before it could perform a task, similar to how students need to study before they can pass an exam.



Moreover, models only read text one way and used static embeddings, which limited their ability to understand human language. Humans use words in different context, and in the syntax of many language context is often given after a target word.

## Why BERT Matters for LLMs

BERT excels at understanding human language because it reads sentences bidirectionally and uses dynamic embeddings. LLMs built on these breakthroughs to create general-purpose models trained on huge corpora. The size of these models and their ability to parse human language well enables them to perform tasks without needing to be specifically trained for them.

This is called **zero-shotting**.



# Using LLMs for Text Analysis

- **Zero-shot:** Giving the model a task without providing examples or training data.  
Example: "Classify this text as positive or negative: [text]"
- **Few-shot:** Provide a handful of examples before task.  
Example: "Here is a positive text: [text]. Here is a negative text: [text]. Now classify this text: [text]"
- **Fine-tune:** Retrain the model on a specific dataset. Mostly used with specialized or domain-specific text corpora (e.g., legal documents).
- **RAG (Retrieval-Augmented Generation):** Have the model retrieve information before task. The model generates response based on information in it retrieved. Used to reduce "model hallucinations" and keep responses grounded in the data, or when specific information is needed not in model training data.



# Natural Language Inference (NLI)

NLI is a classification task where a model evaluates the **logical relationship between two pieces of text**, a statement (premise) and a hypothesis. It produces a probability score (**confidence score**) for each option:



- **Entailment:** The hypothesis matches the statement.
- **Contradiction:** The hypothesis clashes with the statement.
- **Neutral:** The hypothesis is unrelated the statement doesn't provide enough information.

We typically interpret the entailment score as the model's confidence that the hypothesis applies to the statement.



# Encoder vs. Decoder Models

	Encoder	Decoder	Encoder-Decoder
What it does	Read/understand text and generate representation	Generate human-like text from prompt or existing text	Transform input into different output
Used for	Classification, NLI, embeddings, semantic similarity	Chatbots, drafting, text completion, Q&A	Machine translation, summarization, data transformation
Strengths	Deep contextual understanding of text	Fluent, creative, adaptable text generation	Transform input into output, structured tasks
Weaknesses	Cannot generate long-form text	Weak fine-grained contextual understanding, prone to hallucinations	Complex training, high computational cost
Examples	BERT, RoBERTa, DeBERTA	GPT, Claude, Gemini	T5, BART, MarianMT

## Interacting with LLMs: APIs

An **API** (Application Programming Interface) is a programmatic way to access LLMs. We send the model text through code, and it sends back a reply. There is no ChatGPT-style chat window. The model runs quietly in the background.



APIs are great for **automation and for task scaling**. They enable users to process large datasets automatically and reliably.

On the other hand, APIs require **technical setup and expertise**. They are also **not interactive** – you can't "chat" with the model.

## Interacting with LLMs: Chat-Based Interfaces

**Chat-based interfaces**, such as ChatGPT, enable interactive conversations with LLMs. The user and the model exchange typed messages back and forth in a conversational format.



Chat-based interfaces are **intuitive, fast, and don't require technical set up**. They are good for quick, small experiments, brainstorming, and Q&A.

On the other hand, they are **hard to automate and scale** and **prone to "drift"**. Chat-based interfaces also do not allow for controlling output format, meaning answers are often **less consistent** compared to APIs.

## Interacting with LLMs: What to Choose?

- **API:** quiet, large-scale automation requiring consistent and reliable output that you can leave running.
- **Chat:** quick, small tasks; human-in-the-loop tasks; initial exploration, testing, and task design.



## Local vs. Cloud-Based LLMs

### UBC Policy

All UBC researchers must follow UBC's data security and privacy policy:

- Only certain AI models and platforms are permitted for research use.
- Models that send data outside Canada (e.g., some commercial APIs) are restricted.
- Tools like DeepSeek are not approved for handling research data.
- When in doubt, consult UBC's Research Ethics Board or IT guidelines before uploading sensitive material.

Rule of thumb: If your data includes interviews, transcripts, or personal information, you likely need to keep it local or use UBC-approved services.



# Local vs. Cloud-Based LLMs

**Local models:** Model runs on your own system

- Data stays in your own environment, so you have more privacy and control.
- No outside party sees your data or analysis.
- Good for data/output you don't want to give others access to.
- Downside: requires hardware and technical setup.

**Cloud/Host APIs:** Model runs on external platform

- Fast start, easy to use, no or very minimal setup needed.
- Access to state-of-the-art proprietary models.
- Data leaves your system, subject to company policies.
- Pay-per-use can be expensive at scale.



# LLM Ethics

Ethical issues are present in all aspects of working with LLMs. An issue closer to text analysis has to do with **bias**.

- Language is deeply tied to culture, identity, and history.
- Word embedding models learn from textual corpora that are **inherently biased**. Consider how cultural products reflect social biases and stereotypes.
- As a result, LLMs are in themselves biased and may generate text that reinforces and perpetuates stereotypes.
- Bias mitigation strategies may include training models from scratch using carefully curated corpora to fine-tuning pre-trained models.
- However, these strategies can only mitigate, not eliminate bias. **It is therefore important to acknowledge bias in all work with LLMs.**



# Thank you!

UBC Sociology  
Centre for Migration Studies  
Centre for Computational Social Science  
UBC prAxIs

**Tom Einhorn**  
**September 2025**

